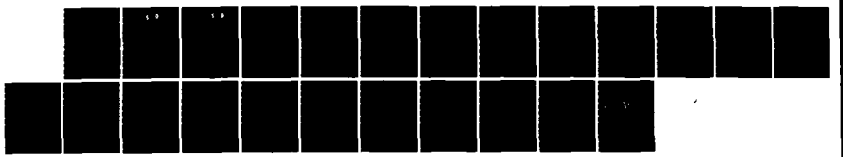
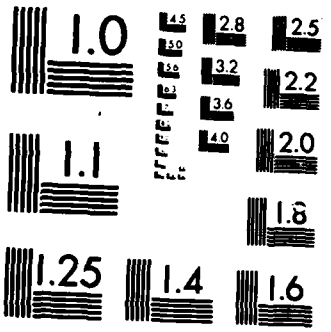
AD-A165 854 THE VOTE TALLYING CHIP: A CUSTOM INTEGRATED CIRCUIT(U) 1/1
ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE
D SHER ET AL. NOV 84 TR-44 N00014-82-K-0193

UNCLASSIFIED F/G 9/5 NL





The Vote Tallying Chip: A Custom Integrated Circuit

David Sher and Avadis Tevanian*
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 144
November 1984

Department of Computer Science
University of Rochester
Rochester, New York 14627

The Vote Tallying Chip: A Custom Integrated Circuit

David Sher and Avadis Tevanian*
Computer Science Department
The University of Rochester
Rochester, New York 14627

TR 144
November 1984

*present address: Computer Science Department,
Carnegie-Mellon University, Pittsburgh, PA 15213

A custom integrated circuit with applications to computer vision has been designed and fabricated. It is the first generation of a series of circuits that can act as specialized hardware for finding maxima in a distribution that is constructed by vote tallying. Many applications in artificial intelligence use peaks of histograms or areas of high density in distributions as the basis for decisions. This series of designs is meant to operate in those domains where histograms are constructed by tallying samples. The current design simply constructs the histogram given a stream of samples. An extension of content addressable memory is used to tally the samples. The design has been made fast by extensive use of pipelining.

This material is based on work supported under a National Science Foundation Graduate Fellowship grant number SPE-8350104. This work was supported in part by NSF Grant MCS-8203028 and a Defense Advanced Research Project Authority grant number N00014-82-K-0193.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 144	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* The Vote Tallying Chip: A Custom Integrated Circuit		5. TYPE OF REPORT & PERIOD COVERED technical report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) David Sher and Avadis Tevanian		8. CONTRACT OR GRANT NUMBER(s) N00014-82-K-0193
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Rochester Rochester, NY 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE November 1984 13. NUMBER OF PAGES 18
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. *(of this report)* unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

VLSI, Hough Transform, Cluster Analysis, Histogram, Artificial Intelligence, Computer Vision, Systollic Array

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A custom integrated circuit with applications to computer vision has been designed and fabricated. It is the first generation of a series of circuits that can act as specialized hardware for finding maxima in a distribution that is constructed by vote tallying. Many applications in artificial intelligence use peaks of histograms or areas of high density in distributions as the basis for decisions. This series of designs is meant to operate in those domains where histograms are constructed by tallying samples. The current design simply constructs the histogram given a stream of samples. An

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. Abstract (cont.)

extension of content addressable memory is used to tally the samples. The design has been made fast by extensive use of pipelining.

1. Introduction

Traditional computer architectures supply particular primitive functions such as add, load, or jump. Artificial intelligence applications demand different primitive functions. Usually these primitives are calculated using software on traditional hardware. This has made most artificial intelligence programs slow and inefficient. Custom VLSI can supply many of the primitives that artificial intelligence applications require.

In computer vision it is common for systems to receive information as votes (that are samples from some distribution). These votes must be tallied to build a histogram. The local maxima of the resulting histogram is usually the statistic of interest. Clusters, which form the basis of traditional pattern recognition, are areas of high density. An area of high density on a continuous distribution becomes a local maxima when the distribution is discretized. Certain approaches to image segmentation use histograms to group related pixels into regions. The hough transform (a popular technique for deriving high level image features from low level ones) is based on local maxima of histograms.

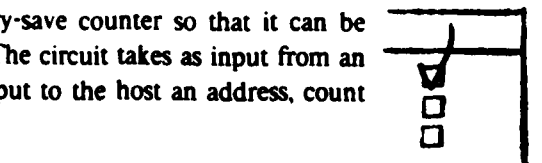
The common alternatives to mode based techniques are based on the least square error. These suffer serious degradation from outliers. The modes are more resistant to outliers.

The most common means of implementing tallying schemes is to have each possible element of the sample space assigned a corresponding word of an array. This word is incremented when a vote is received for that element. After voting the maxima are found in the array. For many applications the array can be multidimensional and require huge amounts of memory if ordinary memory is used. Caching schemes can reduce the amount of memory necessary especially if the voting is sparse in the accumulator array.

Accommodating caching schemes leads to the use of content addressable memory with counters as the basis of the design. Because the votes come into the array in a continuous stream the circuitry that was developed is pipelined. The circuit is laid out as a vector of memory units each of which receives signals only from the adjacent units. This design allows one to cascade chips. Also the time the entire circuit takes to accept a vote is only the time it takes for a unit to process the signal.

The memory units are a combination register and comparator to hold the location in sample space, a counter to store the number of votes for that location, and a finite state machine to decode incoming signals and control the rest. The address register and counter also can take their value from the next unit. This is done when the accumulated tallies are retrieved. The counter is a carry-save counter so that it can be arbitrarily extended without affecting the speed or design of the circuit. The circuit takes as input from an external host two control lines and an address for input. It sends as output to the host an address, count and a signal indicating the validity of the output.

Our current layout explores the effectiveness of the basic concepts. It was fabricated in April 1984, and was tested in June 1984. Five of the six chips fabricated worked perfectly according to the tests. The sixth did not work at all and thus can be considered a fabrication error.



2. Task Description

The task of this hardware is vote tallying. Vote tallying is the process accepting a stream of samples and determining statistics about the resulting distribution. When a president is elected the samples are the votes for the various candidates. The statistic that is being determined is the mode of the distribution. When vote tallying is used in the context of vision the votes arise from easily recognized elements of an image, such as pixels or edges. The statistics, in this context, are the local maxima.

This paper describes a device that take as input a stream of samples from some distribution. The output is the exact histogram of the distribution. Later generations of this chip could generate a modified histogram whose local maxima have a high probability of being local maxima of the original distribution.

2.1. Applications

There are many applications for such a device in computer vision. These applications use the local maxima instead of the mean. Local maxima are unchanged by outliers and certain kinds of systematic noise. Also a distribution can only have one mean but many local maxima. Three such applications are the hough transform, clustering, and segmentation.

2.1.1. The Hough Transformation

The Hough transformation is an algorithm schema that takes advantage of the stability of local maxima under noisy conditions. The Hough transform has been applied to finding shapes in images [Hough62][Ballard81a], finding the velocity from optical flow [Ballard81b] and in many other contexts.

Consider the application of finding shapes in images. Assume an array of directed edges is the input. Each edge in the image can be a part of only a certain subset of the shapes. Thus each edge can be considered evidence for some set of shapes. The Hough transformation takes each edge and transforms it into a set of votes for the shapes it can participate in. These votes once tallied form a distribution. The local maxima of this distribution are the shapes that are supported by the evidence unambiguously thus are the shapes that are likely to be in the image. The chip can be used for the vote tallying stage of this algorithm. When using optical flow for rigid body motion one need only realize that the optical flow at each point of an image can only be evidence for a constrained set of rigid body motions and then apply the same reasoning.

2.1.2. Clusters

A prominent application for cluster analysis is pattern classification [Fukunaga72]. In the "training" phase of pattern recognition it is hoped that samples from the domain for clusters in feature space, with each cluster corresponding to a domain object to be recognized (classified). Later, in use, the pattern recognition consists of deciding to which cluster a sample "really" relongs. The chips described in this paper will be useful during the training phase (finding cluster locations) because a cluster of points in continuous space produces a local mode in a corresponding discrete space.

2.1.3. Segmentation

The visual field often is composed of several objects with different colors or intensities [Ohlander79]. Many of the most successful approaches to image segmentation take advantage of this. Shading, lighting variation, natural variation and other visual distortions will cause each object to appear with non-uniform color. Never the less the colors in various regions should cluster about several specific values. These values can be detected by having each pixel vote for its color into an accumulator array. Once again the chips described here can act as such an accumulator array. Once these cluster points are detected they can be used to create a first approximation towards an image segmentation.

2.2. Advantages of Content Addressable Memory

The hardware makes use of content addressable memory (with some modifications) to store the distribution. This has these advantages:

(1) Only the actual points voted for need be stored rather than the entire address space. In applications requiring multidimensional spaces this is important.

(2) There is a natural pipelined implementation for this task using content addressable memory.

(3) Further space can be saved by periodically flushing out low frequency points in the distribution.

2.3. Advantages of Cache Flushing

Later generations of this hardware will have the capability of flushing out low frequency votes. Results on the efficacy of this technique for getting the modes and maxima of the distribution while often saving more than an order of magnitude of space are documented in [Brown82b] and [Brown83a]. In circuitry featuring heavily pipelined content addressable memory the mechanism of flushing is straightforward.

3. Implementation

The first generation of a series of vote tallying chips has been fabricated and tested. It is a simple framework that can be embellished to produce time and space efficient hardware for detecting local maxima and modes. In this section the framework will be described in increasing detail.

3.1. Serial Implementation

To decide whether a particular algorithm is worthwhile it is necessary to consider if there is another algorithm that implements the same function faster or using less memory or both (depending on what is the main cost in your particular system). In particular if there is a less expensive serial implementation of a function than the proposed parallel implementation then using the parallel implementation is not worthwhile.

The simplest serial algorithm assigns an element of an array to every location in the address space. The collection of a sample increments the element whose address corresponds to the value of the sample. At the end of the sampling the local maxima are discovered. If the address space is multidimensional the size of the array that is allocated and searched can become huge. A hash table can solve many of these problems. Content addressable memory can speed many applications that use hash tables. When a single image can generate 10,000 samples efficiency gains become important. Also flushing and sorting can be done by special purpose hardware while the sampling continues.

3.2. Operation of the Chip

This chip acts as content addressable memory with counters. There are three actions it can perform:

(1) insert sample

(2) retrieve data

(3) reset memory

The chip is implemented by an array of units each of which contain memory to hold a sample and a count. Inserting a sample that has yet to be inserted allocates a unit with a count of 1 to it. If the sample has been seen the unit originally allocated to that sample has its count incremented. The value of the first unit in the memory is always available for probing. The retrieve data signal causes the first unit in memory to be output and deleted. While there are units of memory available the second unit takes the place of the first. In the next clock cycle this signal is then broadcast to the third unit that takes the place of the second and so on till there are no more units with data in them. Thus the retrieve data causes the data in the memory to shift towards the first unit. This can be recognized as a pipelined version of a hardware stack or queue. retrieving data takes two clock cycles. The reset memory operation frees all units for further allocation.

3.3. The Functional Unit

The central concept of this generation of chips is the use of an array of identical devices. Each of these devices is one word of the content addressable memory. The first element is connected through a decoder to the signals coming from the primary pins. The last element is connected through an encoder to the pins for cascading. The layout of the functional units on the chip is shown in Figure 1. In the current circuitry there are four functional units.

3.3.1. The Composition

The functional unit has three parts, a comparator, a counter, and a finite state machine. The comparator stores addresses and compares an incoming sample with that address. The counter stores the number of samples at that address. The finite state machine takes the control signals and sends the correct signals to the comparator and the counter. The finite state machine also has a bit of memory that indicates whether the address stored in the comparator is significant. A labeled functional unit is shown in figure 2.

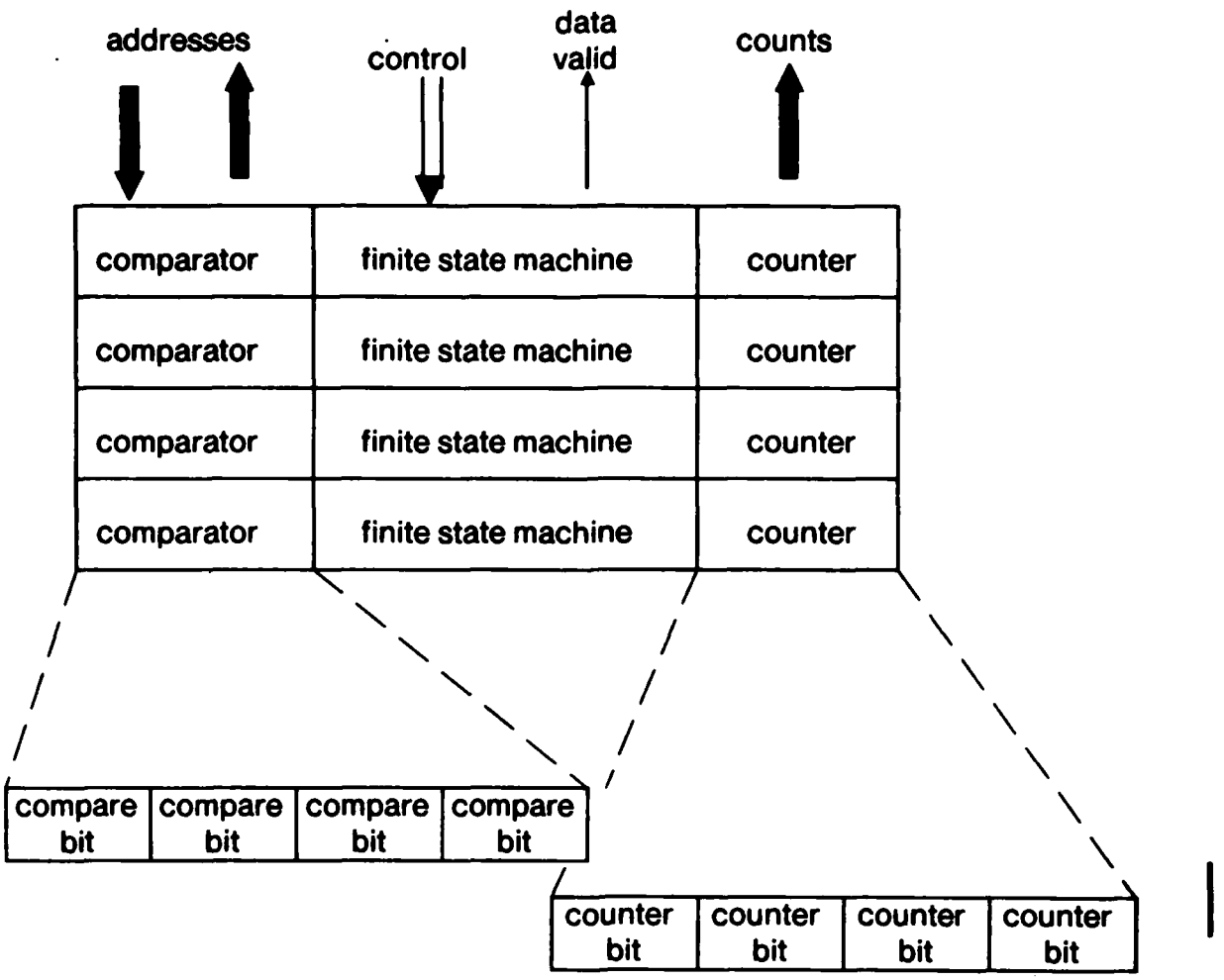


Figure 1: Floor Plan for Chip

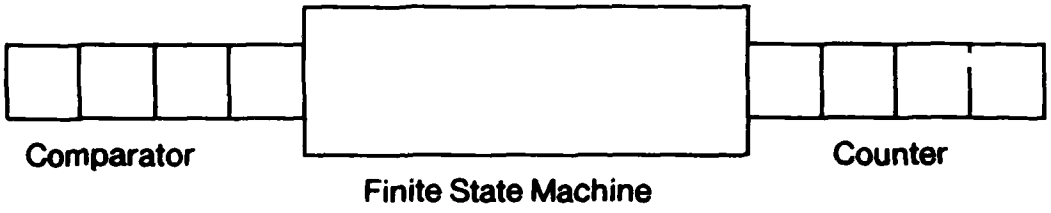


Figure 2: Schematic for Functional Unit

3.3.2. The Comparator

The comparator accepts control signals from its associated finite state machine. It gets signals on φ_2 of the two phase clock. The comparison is done on φ_1 of the two phase clock. The comparison is performed by a single complex gate. The comparison circuitry is on the critical path of the chip. This means that the speed of the chip depends directly on the speed of this circuitry. Figure 3 is a a schematic describing the comparison circuitry.

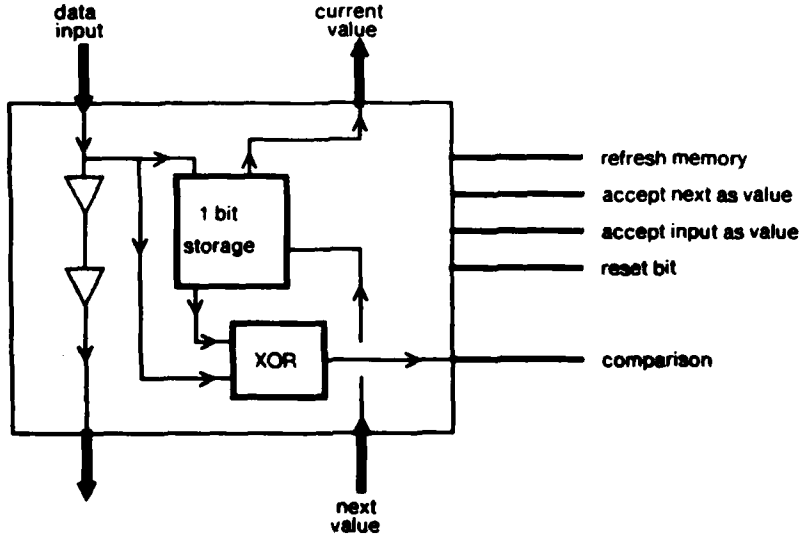


Figure 3: Schematic for One Bit of Comparison Circuitry

3.3.3. The Counter

The counter accepts control from the finite state machine to the left. It gets signals on φ_2 of the two phase clock. It is a carry save counter modified to allow parallel load (for the retrieve data command). To send an increment signal the carry in input of the low order bit is set high. Figure 4 is a schematic description of the counting circuitry.

3.3.4. The Finite State Machine

The finite state machine accepts input on φ_1 and sends output on φ_2. Its input is four control signals and a line that is high when the sample is equal to the comparator's address. It generates the signals to comparator, counter and next unit's finite state machine. The data valid bit is the only bit of state. It was designed as random logic to fit well between the counter and comparator. Figure 5 is a schematic description of the finite state machine's functionality.

3.4. Pipelining

The counters are pipelined using carry-save logic. Another form of pipelining used by this chip is that the signals as they pass from one unit to another are pipelined. This is because the finite state machines take input on φ_1 and output on φ_2. The samples as they pass down the chip are pipelined as well. This means that the clock speed of the chip is the speed of a single unit. The pipelining of the retrieve data command means that it takes two clock cycles to retrieve each piece of data. This slowdown is predicted by the circuit retiming theorem [Leiserson81] or the systolic conversion theorem [Leiserson83].

3.5. Physical Parameters

The first generation of this series was fabricated in 4 micron NMOS. The width of the first generation is 3.326 micron and the length is 2.028 micron. It is perfectly rectangular. The static power consumption is approximately 300 milliwatts. Using the timing simulator tsim, designed by Christopher

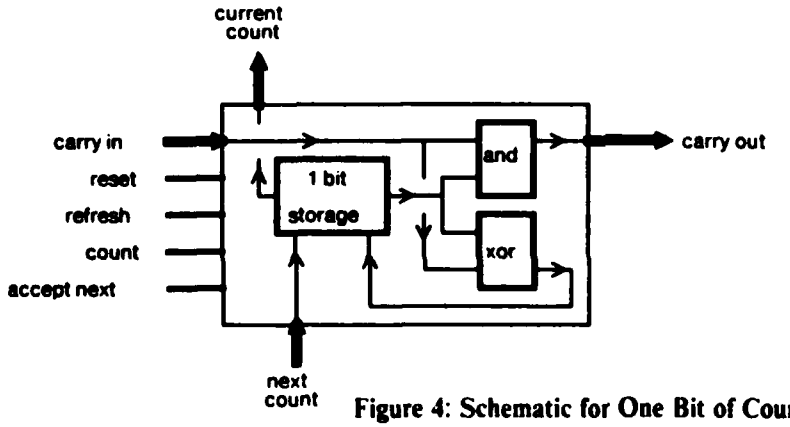


Figure 4: Schematic for One Bit of Counter

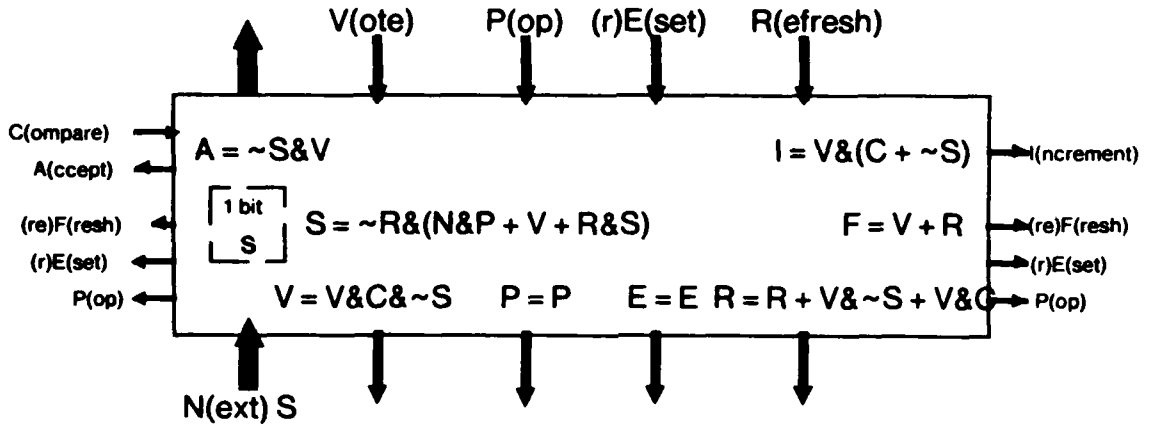


Figure 5: Schematic of Finite State Machine Circuitry

Terman, some conservative estimates for the timing of my chip were generated. The longest delay between clocks seems to be 428.3 nanoseconds in five random test cases. This would indicate a clock cycle of approximately 1 MHz.

3.6. The Finished Product

This chip was designed using the Icarus and Caesar [Ousterhout83] vlsi design editors. It was extensively simulated using esim (a transistor level simulator for digital NMOS, also designed by Christopher Terman) . It was then fabricated by MOSIS (A DARPA funded organization that arranges fabrication of chips and pc-boards whose floor plan is sent to them in a foundry independent format) and the resultant chips tested. Five of the six chips fabricated were found to be functional. This high yield is attributed to the use of conservative design rules and the extensive simulation (and debugging) done before fabrication.

3.6.1. The Simulation

Esim was used in the early design effort to simulate each of the components of the chip as they were designed. When the entire layout was put together a program that cascaded 4 esim simulations of the design was written and used to test the entire chip under realistic (for a prototype) conditions. It also checked that the chips could be cascaded without introducing logical errors. Since the chip consists largely of memory exhaustive testing of even one chip is not a possibility. Semi-random data was generated and input to the simulation. When the design behaved properly for all the runs the chip was deemed ready for fabrication. See Appendix 1 for sample test data.

3.6.2. Results from Fabrication and Testing

In April 1984 the CIF description of the chip was sent to MOSIS for fabrication in 4 micron technology NMOS. In May 1984 the fabricated chips from the first fabrication run were received by the authors of this paper. They were tested at Carnegie Mellon University using "The CMU Test Rig" [Anantharam83]. Five of the six chips fabricated were found to have full functionality. The sixth was unable to accept signals and its outputs were not acceptable for an NMOS device. This was attributed to a fabrication error. Thus the yield for this prototype was found to be 83%. Since speed was not an issue in this design there was no effort to determine the maximum clock rate for the chip.

4. Future Generations

The future generations of chips would make use of an architecture similar to this generation to detect local maxima. Techniques for improving the flexibility and efficiency of these chips are described below.

4.1. Weighted Votes

Weighted votes (or samples) are ordered pairs of sample and amount of evidence. Work has been done with using the hough transform on weighted votes [Brown83a]. To tally weighted votes in this framework one need only replace the counters with full adders. The weights are added in by the full adders to their registers.

4.2. Sorting

Maintaining the data in order of increasing counts has many applications. Detecting maxima is simplified when the data is sorted. This can be done by running the data through sorting hardware after output. It is possible to build a sorter into an array of units. Building in a sorter keeps the votes sorted for the most part while the voting is taking place. This makes certain flushing schemes much easier. Using the result of [Leiserson81] it can be shown that there is a method that ensures that to the host the array always appears sorted. This method requires considerably more hardware (it would consume approximately five times as much area). Using a modified compare and exchange sorting algorithm the sorting can be done with twice as much hardware (as not sorting). Both methods spend two of every three clock cycles sorting. The gains from sorting the array dynamically might justify this cost.

4.3. Flushing

The main motivation for this research was to create an architecture to support cache flushing. Flushing a cache is to remove certain entries in the cache periodically or on demand to keep it from filling up. The hardware described acts as a cache. There is evidence that with a good flushing scheme maxima can be detected with much less memory than without [Brown82b] [Brown83a] A variety of flushing schemes can be built into this hardware. The actual flushing of a unit can be performed simply by having the unit act as if it received a retrieve data command, it deletes its entry and accept the entry of the next unit and tell the next unit to do the same.

The addresses that have accumulated little evidence are good candidates for flushing. These are the units with low values in their count registers. Dynamic sorting would make sure that most of these units were together on the far end of the array. Research has been done on a variety of schemes that use this

idea.

If sorting is not to be performed dynamically then each unit must decide whether to flush based on the number in its counter. One such scheme is to subtract one from the entire cache and then flush all the addresses with zero weight. Several schemes for doing this were proposed in [Brown82b] and examined in [Brown83a].

If the units are dynamically sorted then flushing schemes can use order statistics. One such scheme is to flush the bottom third of the array when full. This would be difficult to do to an unsorted array. It is easy to do to a sorted array.

Schemes for flushing that use a low resolution histogram are being studied [Brown83b]. These scheme involves having a command to flush all the votes in a rectangle received by the chip from the host. A minor change to the comparator and finite state machine is required for such schemes.

5. A System that could Use the Chip

This section describes how this chip could be used in a special purpose vision system. The system that will be described takes digitized images and searches them for a specified set of shapes. This illustrative use is one of many possible uses, and has not been implemented.

5.1. A Short Description of G-Hough

The G-Hough is a technique for recognizing specified shapes in images developed by Dana Ballard [Ballard81a]. It is a fast table-based algorithm for correlation detection [Brown82a]. This technique is used for determining the position orientation and scale of the shapes in an image from the line segments in the image. In this paper a modified algorithm that starts from edges rather than segments will be described.

From the prespecified shape a data structure called an R-table is built. An R-table has an entry for every edge orientation. Each entry contains a set of possible center points for shapes at a particular orientation and size that might contain this edge. G-Hough proceeds by taking the edges of the input image and iterating over the possible orientations and scales for the image. The R-table specifies how each edge in the image should vote for the size and orientation of shapes that are consistent with the image data. This is described more exactly by the pseudocode below.

```
For x = 1 to image_width
   For y = 1 to image_length
        If edge at x,y Do
           For scale = min_scale to max_scale
                For orientation = 0 to 360 step orientation_step
                   With thing in R-table[orientation[x,y]+orientation] Do
                        vote_for((x,y) + thing*scale)
                   End
                End
```

```
        End
      End
    End
  End
```

5.2. The Components of the System

The system is connected to a camera and digitizer. The output of the digitizer is input to an edge detector. Edge detection at video rates is commonplace today. The output of the edge detector is connected to a set of shape recognizers, one for each shape to be recognized, that return good possible positions for the shape. Then a simple post processor checks out these possibilities and returns the top n shapes and their positions. A graphic representation of such a system is shown in Figure 6.

5.3. The Shape Recognizers

The shape recognizers each can detect good possible positions for a single shape. They are composed of two processors. The first processor does the preprocessing to execute the G-Hough developed by Ballard [Ballard81a]. This processor creates a set of votes for input to a vote tallying chip. The output of the vote tallying chip is a set of possible positions for the shapes being detected and the strength of the evidence supporting the existence of the shape at that point. The top m of these can be found using a simple sorter (if the output is not already sorted).

6. Conclusion

In this paper an architecture has been presented that, suitably embellished, can be a useful addition to many systems for computer vision. There are many different applications that can be speeded by specialized hardware of this form. One particular way of using this device has been described. Other statistical tasks that require mode based estimation or detection of local maxima can be speeded by use of this technique. Hardware designed according to this architecture has been successfully fabricated.

Acknowledgements

This work would have been impossible without the advice and help of Chris Brown and Gershon Kedem.

APPENDIX

In this appendix examples of sample input and output of the test routines is given. The input was generated by a program that took text samples and generated votes corresponding to the letters mod 16. Mod 16 was chosen because the chip has a four bit address space. It generated a null signal when it encountered white space. In the input files numbers from 0 to 15 are votes. -1 indicates a null signal. -2 indicates the end of input. Each input file terminates with an end of input. Each output file contains the histogram in binary that is output by the chip in simulation. The first number after "popping:" is the

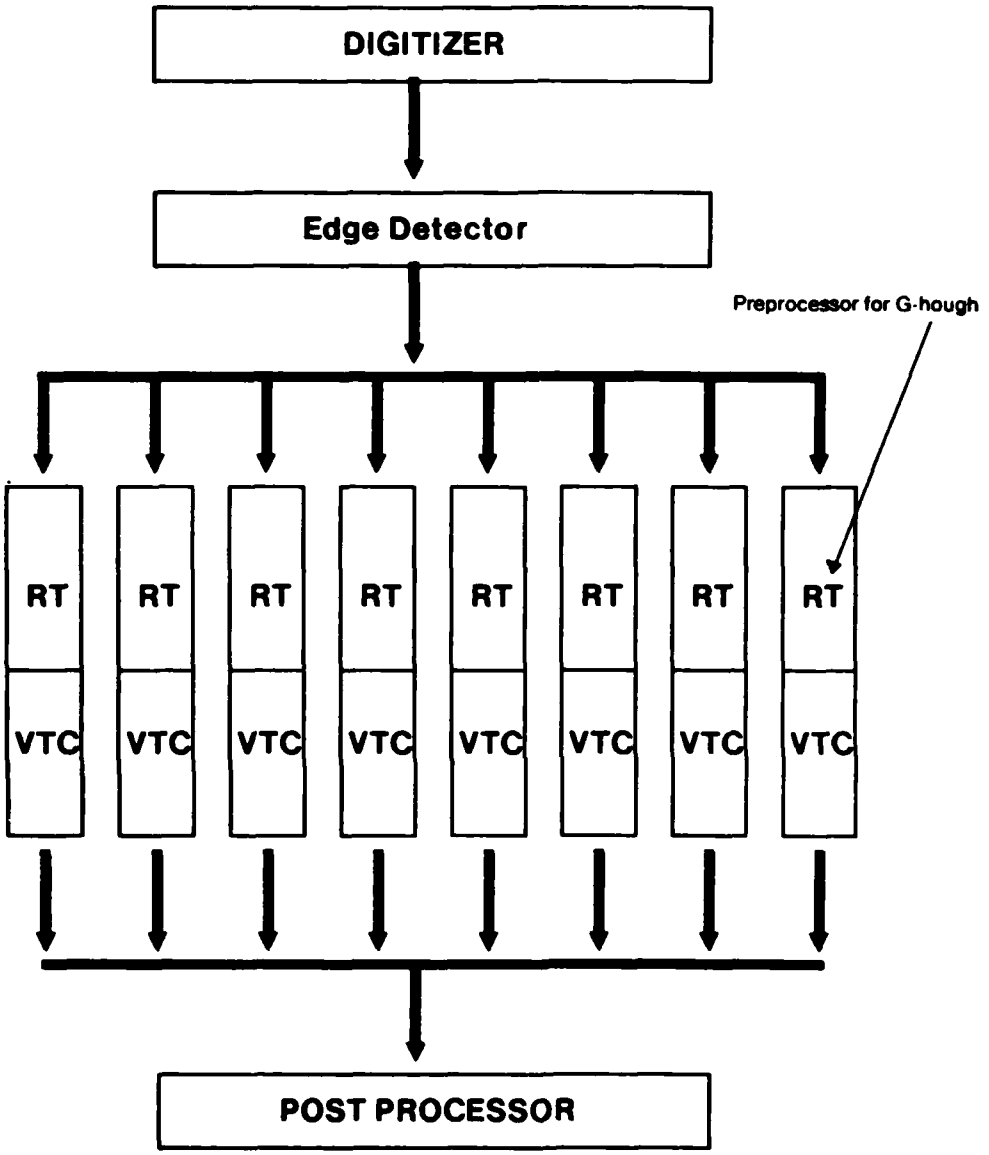


Figure 6: Scheme for Finding Shapes in Images

address. The second number is a bit signaling whether the output is significant. The third number is the count at that address. Each entry is output twice for reasons described in the paper.

Input file 1:

-1 3 9 14 3 12 5 4 5 -1 12 3 4 4 9 15 14 8 14 -1 3 9 14 3 12 5 4 5 -1 2
13 1 3 2 15 3 14 8 2 -1 -1 9 14 4 -1 1 3 3 5 13 11 1 6 13 11 -1 -1 13 1
9 14 8 9 -1 11 -1 -1 2 5 7 9 3 4 5 2 -1 3 8 1 2 -1 3 11 -1 -1 7 8 9 12
5 8 8 15 5 7 8 8 13 1 11 5 6 15 4 5 8 3 -1 13 -1 7 5 4 3 8 1 2 8 9 9 9
9 11 -1 -1 0 2 9 14 4 15 8 15 5 7 8 8 9 11 -1 13 -1 -1 13 1 11 5 6 15 4

5 -2
[0] 1
[1] 8
[2] 8
[3] 14
[4] 10
[5] 15
[6] 3
[7] 5
[8] 15
[9] 14
[10] 0
[11] 8
[12] 4
[13] 8
[14] 8
[15] 7

Output file 1:

popped: 0011 1 1110
popped: 1001 1 1110
popped: 1001 1 1110
popped: 1110 1 1000
popped: 1110 1 1000
popped: 1100 1 0100
popped: 1100 1 0100
popped: 0101 1 1111
popped: 0101 1 1111
popped: 0100 1 1010
popped: 0100 1 1010
popped: 1111 1 0111
popped: 1111 1 0111
popped: 1000 1 1111
popped: 1000 1 1111
popped: 0010 1 1000
popped: 0010 1 1000
popped: 1101 1 1000
popped: 1101 1 1000
popped: 0001 1 1000
popped: 0001 1 1000
popped: 1011 1 1000
popped: 1011 1 1000
popped: 0110 1 0011
popped: 0110 1 0011
popped: 0111 1 0101
popped: 0111 1 0101
popped: 0000 1 0001
popped: 0000 0 0001
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

Input file 2:

4 3 -1 1 -1 2 0 0 -1 4 11 -1 9 -1 0 5 0 1 11 -1 12 -1 14 0 11 -1 2 -1 2
4 -1 2 4 -1 1 2 -1 2 0 11 -1 12 -1 14 4 11 -1 2 -1 8 -1 4 0 -1 1 2 -1 2

0 11 -1 12 -1 14 9 11 -1 2 -1 1 6 -1 2 8 -1 1 2 -1 1 4 11 -1 12 -1 14 2
11 -1 2 -1 2 4 -1 -2
[0] 8
[1] 7
[2] 15
[3] 1
[4] 8
[5] 1
[6] 1
[7] 0
[8] 2
[9] 2
[10] 0
[11] 9
[12] 4
[13] 0
[14] 4
[15] 0

Output file 2:

popped: 0100 1 1000
popped: 0011 1 0001
popped: 0011 1 0001
popped: 0001 1 0111
popped: 0001 1 0111
popped: 0010 1 1111
popped: 0010 1 1111
popped: 0000 1 1000
popped: 0000 1 1000
popped: 1011 1 1001
popped: 1011 1 1001
popped: 1001 1 0010
popped: 1001 1 0010
popped: 0101 1 0001
popped: 0101 1 0001
popped: 1100 1 0100
popped: 1100 1 0100
popped: 1110 1 0100
popped: 1110 1 0100
popped: 1000 1 0010
popped: 1000 1 0010
popped: 0110 1 0001
popped: 0110 0 0001
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

Input file 3:

14 4 9 4 12 5 -1 4 8 5 3 9 3 -1 14 15 4 5 3 -1 3 1 12 -1 -1 15 3 4 15 2

5 2 -1 1 0 12 -1 1 9 8 3 -1 14 3 0 1 3 5 -1 7 -1 -1 14 3 5 14 4 5 2 -1
6 1 9 12 5 2 5 -1 2 5 3 15 6 5 2 9 -1 9 14 -1 4 9 3 4 2 9 2 5 4 5 4 -1
3 15 13 0 5 4 1 4 9 15 14 3 -1 -1 14 3 5 14 4 -2
[0] 3
[1] 6
[2] 8
[3] 14
[4] 13
[5] 15
[6] 2
[7] 1
[8] 2
[9] 9
[10] 0
[11] 0
[12] 4
[13] 1
[14] 9
[15] 6

Output file 3:

popped: 1110 1 1001
popped: 0100 1 1101
popped: 0100 1 1101
popped: 1001 1 1001
popped: 1001 1 1001
popped: 1100 1 0100
popped: 1100 1 0100
popped: 0101 1 1111
popped: 0101 1 1111
popped: 1000 1 0010
popped: 1000 1 0010
popped: 0011 1 1110
popped: 0011 1 1110
popped: 1111 1 0110
popped: 1111 1 0110
popped: 0001 1 0110
popped: 0001 1 0110
popped: 0010 1 1000
popped: 0010 1 1000
popped: 0000 1 0011
popped: 0000 1 0011
popped: 0111 1 0001
popped: 0111 1 0001
popped: 0110 1 0010
popped: 0110 1 0010
popped: 1101 1 0001
popped: 1101 0 0001
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

Input file 4:

-1 -1 -1 1 3 3 9 9 8 7 9 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 5 14 9 8
-1 0 2 15 7 2 1 13 13 5 2 7 3 -1 13 1 14 5 1 12 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 1 3 3 9 9 8 7 9 -1 -1 -1 -1 14 1 13 5 -1 -1 -1 -1 -1 -1
1 3 3 9 9 -1 13 -1 13 1 0 -1 15 6 -1 1 3 3 9 9 -1 3 8 1 2 1 3 4 5 2 -1
3 5 4 -1 -1 3 9 14 15 0 3 9 3 -1 -1 -1 -1 -1 -1 -1 -2
[0] 3
[1] 11
[2] 5
[3] 15
[4] 2
[5] 6
[6] 1
[7] 4
[8] 4
[9] 13
[10] 0
[11] 0
[12] 1
[13] 6
[14] 4
[15] 3

Output file 4:

popped: 0001 1 1011
popped: 0011 1 1111
popped: 0011 1 1111
popped: 1001 1 1101
popped: 1001 1 1101
popped: 1000 1 0100
popped: 1000 1 0100
popped: 0111 1 0100
popped: 0111 1 0100
popped: 0101 1 0110
popped: 0101 1 0110
popped: 1110 1 0100
popped: 1110 1 0100
popped: 0000 1 0011
popped: 0000 1 0011
popped: 0010 1 0101
popped: 0010 1 0101
popped: 1111 1 0011
popped: 1111 1 0011
popped: 1101 1 0110
popped: 1101 1 0110
popped: 1100 1 0001
popped: 1100 1 0001
popped: 0110 1 0001
popped: 0110 1 0001
popped: 0100 1 0010
popped: 0100 0 0010
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

popped: 0000 0 0000
Input file 5:
-1 -1 -1 -1 -1 -1 -1 2 5 0 15 2 4 -1 15 14 -1 4 8 5 -1 3 4 1 4 5 3 -1
15 6 -1 4 8 5 -1 9 14 6 5 2 3 5 -1 4 9 14 1 13 9 3 3 -1 13 15 4 5 12 5
-1 15 6 -1 4 8 5 -1 2 15 2 15 4 -1 3 9 13 5 12 1 4 15 2 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 1 6 9
4 -1 3 8 5 2 -1 -1 -1 -1 15 8 1 14 -1 -1 15 8 9 15 8 14 15 8 4 15 8 2
15 8 15 -2
[0] 1
[1] 5
[2] 8
[3] 7
[4] 13
[5] 11
[6] 4
[7] 0
[8] 10
[9] 6
[10] 0
[11] 0
[12] 2
[13] 3
[14] 5
[15] 15
Output file 5:
popped: 0010 1 1000
popped: 0101 1 1011
popped: 0101 1 1011
popped: 0000 1 0001
popped: 0000 1 0001
popped: 1111 1 1111
popped: 1111 1 1111
popped: 0100 1 1101
popped: 0100 1 1101
popped: 1110 1 0101
popped: 1110 1 0101
popped: 1000 1 1010
popped: 1000 1 1010
popped: 0011 1 0111
popped: 0011 1 0111
popped: 0001 1 0101
popped: 0001 1 0101
popped: 0110 1 0100
popped: 0110 1 0100
popped: 1001 1 0110
popped: 1001 1 0110
popped: 1101 1 0011
popped: 1101 1 0011
popped: 1100 1 0010
popped: 1100 0 0010
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000
popped: 0000 0 0000

References

Anantharam83.

T. Anantharam, M. Annaratone, R. Bisiani, A. Gupta, C. Ebeling, E. Frank and O. Zajicek, The CMU Test Rig, 147, Carnegie Mellon Computer Science Department VLSI group, November 1983.

Ballard81a.

D. H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, *Pattern Recognition 13*, 2 (1981), 111-122.

Ballard81b.

D. H. Ballard and O. A. Kimball, Rigid Body Motion from Depth and Optical Flow, 70, Department of COmputer Science, University of Rochester, November 1981.

Brown82a.

C. M. Brown, Bias and Noise in the Hough Transform 1: theory, 105, Department of Computer Science, University of Rochester, June 1982.

Brown82b.

C. M. Brown and D. B. Sher, Hough Transformation into Cache Accumulators: considerations and simulations, 114, Department of Computer Science, University of Rochester, August 1982.

Brown83a.

C. M. Brown, M. B. Curtiss and D. B. Sher, Advanced Hough Transform Implementations, 113, Department of Computer Science, University of Rochester, March 1983.

Brown83b.

C. M. Brown, Hierarchical Cache Accumulators for Sequential Mode Estimation, 125, Department of Computer Science, University of Rochester, July 1983.

Fukunaga72.

K. Fukunaga, *Introduction to Statistical Pattgern Recognition*, Academic Press, New York, 1972.

Hough62.

P. V. C. Hough, Method and means for recognizing complex patterns, U.S. Patent 3,069,654, 1962.

Leiserson81.

C. E. Leiserson and J. B. Saxe, Optimizing Synchronous Circuits, *22nd IEEE symposium on Foundations of Computer Science*, , 1981, 22-36.

Leiserson83.

C. E. Leiserson, *Area-efficient VLSI computation*, M. I. T. Press, Cambridge, MA, Oct 1983.

Ohlander79.

R. Ohlander, K. Price and D. R. Reddy, Picture Segmentation using a Recursive Region Splitting Method, *CGIP 8*, 3 (1979), .

Ousterhout83.

J. Ousterhout, *Editing VLSI Circuits with Caesar*, Computer Science Division, Electrical Engineering and Computer Sciences, University of California, Berkeley California, Febuary 1983.

END
FILMED
4-86
DTIC